Final: Incompressible, Laminar Flow over a Rectangular Cavity

John Karasinski
Graduate Student Researcher
Center for Human/Robotics/Vehicle Integration and Performance
Department of Mechanical and Aerospace Engineering
University of California
Davis, California 95616
Email: karasinski@ucdavis.edu

1 Problem Description

Forty five years ago, Mehta and Lavan published a paper on the numerical investigation of flow over a rectangular cavity at low Reynolds numbers [1]. This relatively simple geometry provides tremendous insight into the physics of flow separation, an important flow feature in many applications. A numerical 2D planar model of these incompressible, laminar flows is developed here. In particular, the predicted flow structure (streamline pattern, eddies) and velocity profiles are investigated for a variety of aspect ratios (AR) and Reynolds numbers (Re).

2 Numerical Solution Approach

The icoFoam solver from OpenFOAM 2.3.0 was used to model the solution of this problem. icoFoam is a transient solver for incompressible, laminar flow of Newtonian fluid. Five cases were investigated: a base case with AR=0.5 and Reynolds number (Re) of 100, and additional cases of AR=0.5 with Re=1 and 2000 for AR=0.5, and AR=2.0 and 5.0 for Re=100. A Python script was created to generate the initial conditions and geometry for each case.

The solver is initialized with the initial conditions described in Table 1. Additionally, the boundary field for the inlet and outlet boundaries are set to "zeroGradient" for all of the initial fields and, the boundary field for frontAndBack is set to "empty" for all initial fields, turning this into a 2D problem. The geometry for this problem consists of a channel of width 10 m and height 1 m. A cavity is placed just below the channel and has a width of 1 m and a depth of AR (see Figure 1).

The script also generated the nonuniform mesh for this geometry using the blockMesh tool. This mesh was divided into four regions: the left half of the channel, the right half of the channel, the center of the channel, and the cavity. For the base case, both the left and right halves of the channel were split into grids of 50x50 in the x, y direction, and also used "simpleGrading" to grade the meshes to make them denser near the center of the domain. The center of the channel was also split into grids of 50x50 in the x, y direction. The cavity for the base case was split into grids of 50x25 in the x, y direction and created a uniform mesh. A generalized version of this mesh is available in Table 2.

	internal	lid	fixedWalls
U [m/s]	(0 0 0)*	(1 0 0)*	(0 0 0)*
p [m^2/s^2]	0*	zeroGradient	zeroGradient

Table 1: Initial conditions for simulation (*: uniform field)

Name	x	y	simpleGrading
Left channel	M	M	(0.1 1 1)
Right channel	M	M	(10 1 1)
Central channel	M	M	(1 1 1)
Cavity	M	M*AR	(1 1 1)

Table 2: Mesh configuration algorithm (M=50, AR=0.5 for base case)

The resultant mesh for the base case can be see in Figure 2. The domain of the problem was then split on to four CPUs using the decomposePar tool, and the icoFoam solver was called with the MPI option. The solver than solved the system for 30 seconds to insure convergence and reconstructed the domain using the reconstructPar tool.

3 Results Discussion

4 Conclusion

References

[1] Mehta, U. B., and Lavan, Z., 1969. "Flow in a two-dimensional channel with a rectangular cavity". *Journal*

Fig. 1: Geometry for the AR=0.5 cases

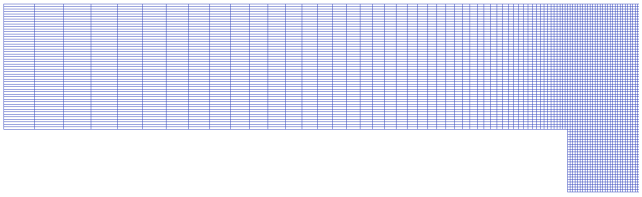

Fig. 2: Closeup on the left half of the mesh generated for the AR=0.5 cases, showing the gradient towards the center

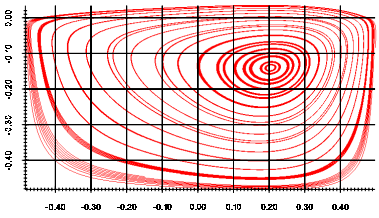

Fig. 3: AR=0.5, Re=2000

of Applied Mechanics, **36**(4), pp. 897–901.

Appendix A: Python Code

```
import subprocess
import os


def inplace_change(filename, old_string, new_string):
    with open(filename) as f:
        s = f.read()

    if old_string in s:
        # print 'Changing "{old_string}" to "{new_string}"'.format(**locals())
        s = s.replace(old_string, new_string)
        with open(filename, 'w') as f:
            f.write(s)
    # else:
        # print 'No occurances of "{old_string}" found.'.format(**locals())


def subprocess_cmd(command):
    process = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
    proc_stdout = process.communicate()[0].strip()
    # print proc_stdout


def generate_folders(ARs, Res):
    for AR, Re in zip(ARs, Res):
        run = "Run" + str(AR) + '-' + str(Re)
        if not os.path.exists(run):
            command = "cp -rf base/ " + run + "/; "
            subprocess_cmd(command)

    print ('Folders generated.')


def create_mesh_file(path, AR, Re):
    M = 100.

    YMESH       = str(int(M * AR))
    INV_GRADING = str(0.1)
    GRADING     = str(10.)
    MESH        = str(int(M))
    AR          = str(-AR)

    inplace_change(path, 'AR',                  AR)
    inplace_change(path, 'XMESH',             MESH)
    inplace_change(path, 'YMESH',            YMESH)
    inplace_change(path, 'INV_GRADING', INV_GRADING)
    inplace_change(path, 'GRADING',         GRADING)
    inplace_change(path, 'MESH',              MESH)


def create_properties_files(path, AR, Re):
    d = 1.           # characteristic size of domain
    NU = str(d / Re)

    inplace_change(path, 'NU_VAR', NU)


def update_dimensions(ARs, Res):
    for AR, Re in zip(ARs, Res):
        run = "Run" + str(AR) + '-' + str(Re)
        path = run + '/constant/polyMesh/blockMeshDict'
        create_mesh_file(path, AR, Re)

        path = run + '/constant/transportProperties'
        create_properties_files(path, AR, Re)

    print ('Config generated.')
```

```
68
69
70  def run_simulations(ARs, Res):
71      for AR, Re in zip(ARs, Res):
72          run = "Run" + str(AR) + '-' + str(Re)
73          if not os.path.exists(run + '/log'):
74              print(run + ' running now.')
75              command = "hdiutil attach -quiet -mountpoint $HOME/OpenFOAM OpenFOAM.sparsebundle; "
76              command += "sleep 1; "
77              command += "source $HOME/OpenFOAM/OpenFOAM-2.3.0/etc/bashrc; "
78              command += "cd " + run + "; "
79              command += "blockMesh; "
80              command += "decomposePar; "
81              command += "mpirun -np 4 icoFoam -parallel > log; "
82              command += "reconstructPar; "
83              command += "streamFunction; "
84
85              subprocess_cmd(command)
86          print(run + ' complete.')
87
88      print('Simulations complete.')
89
90
91  def main(ARs, Res):
92      print('Running ARs ' + str(ARs) + ' with Res ' + str(Res) + '.')
93      generate_folders(ARs, Res)
94      update_dimensions(ARs, Res)
95      run_simulations(ARs, Res)
96      print('Done!')
97
98  if __name__ == "__main__":
99      # Base case
100     ARs = [  0.5]
101     Res = [100.0]
102     #          o                      Broken=x Working=o
103     main(ARs, Res)
104
105     # Additional cases
106     ARs = [0.5,    0.5,   2.0,   5.0]
107     Res = [1.0, 2000.0, 100.0, 100.0]
108     #        x       o      o      o  Broken=x Working=o
109     main(ARs, Res)
```

Listing 1: Code to create solutions